REMARKS

Claims 1-25 are pending in the above-referenced patent application. Claims 1, 15, and 22 are independent. Claims 6, 7, 9, 10, 12-14 and 21 have been deemed allowable.

The Examiner rejected claims 1, 15, 16, 20, 22, and 24 under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 4,868,735 to Moller in view of U.S. Patent No. 5,872,963 to Bitar.

Applicant's independent claim 1 recites a method including directing the processor having a plurality of microengines to swap a currently running context, corresponding to a first thread, in a specified microengine to let another context, corresponding to a different thread, execute in that microengine and cause a different context and associated program counter to be selected.

With respect to independent claims 1, 15, 22, and 24, which the examiner rejected under §103(a) using Moller and Bitar, the examiner argued:

> **Moller did not specifically teach the use of the threads as claimed. Instead, Moller taught the use of loops. However, Bitar disclosed that a suitable number of threads were used to execute a loop in parallel (see col. 10, lines 51-59). Therefore, it would have been obvious to one of ordinary skill in the art to use Bitar in Moller for including the threads as claimed because the use of Bitar could provide Moller the ability of the system to adapt to scientific processing structure at a predefined set of software constructs (e.g., threads), and thereby, minimizing the overall latency of the loop processing in Moller, and because Moller also taught his system was used as building blocks in an architecture divided in control subsections (see col. 1, lines 24-35), ...[pages 3-4 of the Office Action]**

Applicant disagrees.

Moller describes a microprogram sequence controller that includes instruction decode circuitry to allow the controller to process microinstructions (col. 2, lines 63-66). As noted by the examiner, Moller describes a SWAP microinstruction which causes a CMUXCTL signal to be generated to cause multiplexer C-MUX 118 to pass the top of the LIFO stack 136 (LIFO stack 136 facilitates interrupt handling functions) to a down counter 120 (col. 30, lines 39-44). The down counter 120 is used to provide looping control through a microinstruction sequence, and to cause branching from the loop when the down counter 120 reaches one (col. 10, lines 2-5). Execution of the SWAP microinstruction causes an iteration count for an invoking loop to

replace a current iteration count for an invoked loop. In other words, execution of the SWAP microinstruction suspends execution of one loop so that another loop can begin executing. Additionally, with respect to the LIFO stack 136, Moller notes that:

> **This stack maintenance [of LIFO 136] is mandated by the possibility of nested loops, each with its own starting instruction address. As execution passes out of the scope of an inner loop, the starting address of the next-most-inner loop must be on the top of the LIFO stack memory 136. (col. 24, lines 61-66)**

Thus, Moller uses the LIFO stack 136 and the SWAP microinstruction to implement nested loop microprogramming constructs. At no point, however, does Moller disclose or suggest parallel execution of loops is used or even feasible. In addition, Moller neither discloses nor suggests that parallel execution of loops is in anyway necessary or desirable.

Bitar, on the other hand, describes a system and method for context switching between a first and a second execution entity (such as a thread) without having to enter into protected kernel mode (Abstract). Bitar further describes that the threads may be preempted (i.e., interrupted) and have their contexts saved in a designated save area (col. 8, lines 2-5). Bitar explains that one use of multiple preempted threads is to execute parallel loop sections in applications that consist of serial sections and parallel loop sections (col. 10, lines 49-51). As described in Bitar:

> **Transitions from a parallel loop to a serial section may be synchronized by a barrier. At the beginning of each of the parallel loops, the user-level scheduler sets the number requested variable to indicate the desired degree of parallelism. It subsequently queries the number allocated variable to determine the actual degree of parallelism that can be achieved. Based on the number of allocated processors, a suitable number of threads 24 are used to execute the loop in parallel. After the allocated loop work is computed, each thread synchronizes at the barrier; after all the threads have arrived and synchronized at the barrier, the next serial or parallel section is started. (col. 10, lines 51-62)**

Bitar uses threads to enable execution of loops in parallel when an application includes parallel loop sections.

Since Moller microprogram sequence controller and corresponding microinstruction set only supports the execution of nested microprogram loops, but not of parallel loops, to modify Moller's teachings to include Bitar's use of threads to execute loops in parallel would require a

substantial reconstruction and redesign of Moller's system. Applicant thus contends that there is no motivation present in these references to suggest the desirability of their combination.

The examiner's proffered motivation to "provide Moller the ability of the system to adapt to scientific processing structure at a predefined set of software constructs (e.g., threads), and thereby, minimizing the overall latency of the loop processing in Moller, and because Moller also taught his system was used as building blocks in an architecture divided in control subsections (see col. 1, lines 24-35)," fails to recognize that Moller's nested-loop approach is structurally and conceptually incompatible with Bitar's technique to execute loops in parallel. A person of ordinary skill in the art would not be motivated to combine Bitar with Moller since their respective execution designs are entirely at odds with each other and the examiner has not offered a cogent explanation of how one would deal with these incompatibilities. Thus, a person of ordinary skill in the art would lack the motivation to use Bitar's threads, which are used to facilitate Bitar's execution of loops in parallel, to implement Moller's nested-loop approach.

Rejected claims 2-5, 8, and 11 depend from independent claim 1 and are thus patentable for at least the same reasons as independent claim 1. Rejected claims 16-20, which depend from independent claim 15, are patentable for at least the same reasons as claim 15. Rejected claim 23, which depends from independent claim 22, is patentable for at least the same reasons as claim 22. Rejected claim 25, which depends from independent claim 24, is patentable for at least the same reasons as claim 24.

In addition, as noted above, the examiner has rejected claims 2, 23, and 25 under 35 U.S.C. §103 (a) over Moller in view of Bitar , and further in view of Adkins.

Applicant's claim 2 recites "wherein the directing the processor wakes up the swapped out context when a signal specified in a context-swap program instruction is activated." Thus, the swapping operation is performed in accordance with a signal specified in a context-swap program instruction.

As previously explained, Moller discloses a SWAP microinstruction (Moller's col. 30, line 40). As Moller explains, "[e]ach 'machine' instruction is implemented on the microprocessor by a sequence of microinstructions selected by the microprogram sequence controller 10" (emphasis added, col. 4, lines 50-53). Moller's SWAP microinstruction merely causes certain control lines to be asserted (e.g., CMUXCTL signal) that in turn cause Moller's controller to perform certain operations such as passing the content of LIFO 136 to the down

counter 120. Moller's microinstruction, however, is not a program instruction capable of being executed on a CPU-type device, and thus Moller's microinstruction is different from any of applicant's program instructions, including applicant's context-swap program instruction.

Moreover, unlike applicant's context-swap program instruction, Moller's SWAP microinstructions does not include or specify any parameters or signals. Particularly, and as admitted by the examiner on page 5, lines 2-3 of the Office Action, Moller's microinstruction does not include or specify a signal that when activated wakes up a swapped out context.

Accordingly, Moller does not disclose or suggest "wherein the directing the processor wakes up the swapped out context when a signal specified in a context-swap program instruction is activated," as required by applicant's claim 2.

As noted above, Bitar describes context switching between threads. Bitar explains that "[t]o switch context, an execution entity such as a thread, while in user mode, writes the user state of the first execution entity to memory. It then restores the user state of the second execution entity by writing register values associated with the second execution entity to all but a first register and writing the context identifier value to a context identifier location" (col. 5, lines 45-52). But nowhere does Bitar disclose use of a context-swap program instruction to swap contexts. Bitar also does not disclose anywhere waking up swapped-out context, and certainly Bitar does not describe using signal specified in a context-swap program instruction to wake-up swapped-out instruction. Thus, Bitar does not disclose or suggest "wherein the directing the processor wakes up the swapped out context when a signal specified in a context-swap program instruction is activated," as required by applicant's claim 2.

Adkins describes a scheduler executing on a serial communications adapter that schedules tasks at different priority levels (Abstract). While Adkins describes that a sleeping task can be awakened and have its context restored when a new event associated with that task arrives at the port response queue (col. 8, lines 36-57), nowhere does Adkins describe that waking up a context is achieved when a signal, specified in a program instruction, is activated. Indeed, Adkins does not describe use of program instructions to control swapping. Thus, Adkins does not disclose or suggest "wherein the directing the processor wakes up the swapped out context when a signal specified in a context-swap program instruction is activated," as required by applicant's claim 2.

Since none of the references cited by the examiner discloses or suggests, alone or in combination, the feature of "wherein the directing the processor wakes up the swapped out context when a signal specified in a context-swap program instruction is activated," applicant's claim 2 is patentable over the cited art.

Rejected claims 3-5, 8, and 11, which depend from claim 2, are patentable for at least the same reasons as claim 2.

Claims 23 and 25 recite "the context swap instruction wakes up the swapped out context when a specified signal is activated," or similar language. For similar reasons as those provided with respect to claim 2, at least this feature is not disclosed by the art. Therefore, claims 23 and 25 are patentable over the cited art.

It is believed that all the rejections and/or objections raised by the examiner have been addressed.

All of the dependent claims are patentable for at least the reasons for which the claims on which they depend are patentable.

Canceled claims, if any, have been canceled without prejudice or disclaimer.
Any circumstance in which the applicant has (a) addressed certain comments of the examiner does not mean that the applicant concedes other comments of the examiner, (b) made arguments for the patentability of some claims does not mean that there are not other good reasons for patentability of those claims and other claims, or (c) amended or canceled a claim does not mean that the applicant concedes any of the examiner's positions with respect to that claim or other claims.

No fee is believed due. Please apply any charges or credits to deposit account 06-1050, referencing attorney docket 10559-303US1.

Respectfully submitted,

Date: Jan. 31, 2006

Ido Rabinovitch
Attorney for Intel Corporation
Reg. No. L0080

Fish & Richardson P.C.
Telephone: (617) 542-5070
Facsimile: (617) 542-8906

21256965.doc